Pentium® II Xeon™ Processor Specification Update

Release Date: June 1998

Order Number: 243776-001

The Pentium® II Xeon™ processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
June 1998	-001	This document is the first Specification Update for the Pentium® II Xeon™ processor.

PREFACE

This document is an update to the specifications contained in the *Pentium® II Xeon™ Processor at 400 MHz* datasheet (Order Number 243770) and the *Intel Architecture Software Developer's Manual, Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs, Specification Changes, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

S-Specs are exceptions to the published specifications, and apply only to the units assembled under that s-spec.
Specification Changes are modifications to the current published specifications for the Pentium II Xeon processor. These changes will be incorporated in the next release of the specifications.
Errata are design defects or errors. Errata may cause the Pentium® II Xeon™ processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor stepping must assume that all errata documented for that processor stepping are present on all devices.
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.
Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Identification Information

The Pentium II Xeon processor can be identified by the following values:

Family[1]	400-MHz Pentium® II Xeon™ Processor[2]
0110	0101

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after Reset, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after Reset, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Pentium II Xeon processor's second level (L2) cache size can be determined by the following register contents:

512-Kbyte Unified L2 Cache[1]	43h
1-Mbyte Unified L2 Cache[1]	44h
2-Mbyte Unified L2 Cache[1]	45h

NOTE:

1. For the Pentium® II Xeon™ processor, the unified L2 cache size corresponds to a token in the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Specification Update for Pentium® II Xeon™ Processors

GENERAL INFORMATION

Pentium® II Xeon™ Processor Markings

TBD

Pentium® II Xeon™ Processor Identification and Package Information

QDF/ S-Spec Number	Core Stepping	CPUID	Speed (MHz)	L2 Size (Kbytes)	GC82459AA (C6C) Stepping	Processor Substrate Revision	Cartridge Revision	Notes
SL2RH	B0	0652h	400	512	A3	512K-pB	2.0	1, 2
SL2NB	B0	0652h	400	1024	A3	1M/2M-pF	2.0	1, 2

NOTES:

1. These processors will not shut down automatically on assertion of THERMTRIP#.
2. These processors are affected by Erratum 30.

Summary Table of Changes

The following table indicates the Errata which apply to Pentium II Xeon processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
SUB:	This column refers to errata on the Pentium® II Xeon™ processor's substrate or components other than the processor core.
Shaded:	This erratum is either new or modified from the previous version of the document.

NO.	B0	SUB	Plans	ERRATA
1	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
2	X		NoFix	Differences exist in debug exception reporting
3	X		NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in 2-way MP systems
4	X		NoFix	Code fetch matching disabled debug register may cause debug exception
5	X		NoFix	Double ECC error on read may result in BINIT#
6	X		NoFix	FP inexact-result exception flag may not be set
7	X		NoFix	BTM for SMI will contain incorrect FROM EIP
8	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
9	X		NoFix	Branch traps do not function if BTMs are also enabled
10	X		NoFix	Checker BIST failure in FRC mode not signaled
11	X		NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
12	X		NoFix	Machine check exception handler may not always execute successfully
13	X		NoFix	LBER may be corrupted after some events
14	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
15	X		NoFix	A20M# may be inverted after returning from SMM and Reset
16	X		NoFix	Near CALL to ESP creates unexpected EIP address
17	X		NoFix	Mixed cacheability of lock variables is problematic in MP systems
18	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
19	X		NoFix	Memory Type field undefined for non-memory bus transactions
20	X		NoFix	Infinite snoop stall during L2 initialization of MP systems

NO.	B0	SUB	Plans	ERRATA
21	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
22	X		NoFix	Premature execution of load operation prior to exception handler invocation
23	X		NoFix	EFLAGS discrepancy on page fault after multiprocessor TLB shootdown
24	X		NoFix	Read portion of RMW instruction may execute twice
25	X		Fix	Test pin must be high during power up
26		X	Fixed	Processor Information ROM uses WP pin
27	X		Fix	Intervening writeback may occur during split-lock
28	X		NoFix	MC2_STATUS MSR has model-specific error code and Machine Check Architecture error code reversed
29	X		Fix	Cache access while changing BBL_CR_CTL3 configuration may cause hang
30		X	Fix	Thermal sensor may assert SMBALERT# incorrectly
31	X		NoFix	MOVD following zeroing instruction can cause incorrect result
32	X		NoFix	Top 4 PAT entries not usable with Mode B or Mode C paging
33	X		NoFix	MOV with debug register causes debug exception
1AP	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
2AP	X		NoFix	2-way MP systems may hang due to catastrophic errors during BSP determination
3AP	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

NO.	B0	SUB	Plans	SPECIFICATION CLARIFICATIONS
1	X		Doc	Writes to WC memory
2	X		Doc	Multiple processor protocol and restrictions
3	X		Doc	Critical sequence of events during a page fault exception
4	X		Doc	Performance-Monitoring Counter issues
5	X		Doc	POP[ESP] with 16-bit stack size
6	X		Doc	Preventing caching
7	X		Doc	Paging must be enabled before enabling the page global enable bit

NO.	B0	SUB	Plans	DOCUMENTATION CHANGES
1	X		Doc	Invalid arithmetic operations and masked responses to them relative to FIST/FISTP instruction
2	X		Doc	FIDIV/FIDIVR *m16int* description
3	X		Doc	PUSH does not pad with zeros
4	X		Doc	DR7, bit 10 is reserved
5	X		Doc	Additional states that are not automatically saved and restored
6	X		Doc	Cache and TLB description correction
7	X		Doc	SMRAM state save map contains documentation errors
8	X		Doc	OF and DF of the EFLAGS register are mislabeled as system flags
9	X		Doc	CS:EIP pushed onto stack prior to code segment limit check
10	X		Doc	Corrections to opcode maps
11	X		Doc	MP initialization protocol algorithm correction
12	X		Doc	Interrupt 13-General Protection exception (#GP)

ERRATA

1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last non-control floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved in 32-bit mode, the subtraction routine used to calculate the FP Data Operand Pointer will assume the floating-point access was in 32-bit mode, and the high word of the address will be FFFFh instead of 0000h.
IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load which wraps the 64-Kbyte boundary is executed.
- The operating system uses a 32-bit handler on an unmasked exception which occurs during the load.
- The exception handler uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.
WORKAROUND: If the FP Data Operand Pointer is used in a 32-bit exception handler in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium II Xeon processor, as described below:

CASE 1:
The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The Pentium processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the Pentium processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium II Xeon processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:
In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:
If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:
If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

CASE 5:
When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.
IMPLICATION: When debugging or when developing debuggers for a Pentium II Xeon processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.
WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in 2-way MP Systems

PROBLEM: In a 2-way MP system, if an application processor is waiting for a startup inter-processor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

IMPLICATION: After the 2-way MP initialization protocol, only one processor becomes the bootstrap processor (BSP). The other processor becomes a slave application processor (AP). After losing the BSP arbitration, the AP goes into a wait loop, waiting for a STARTUP_IPI.

The BSP can wake up the AP to perform some tasks with a STARTUP_IPI, and then put it back to sleep with an initialization inter-processor interrupt (INIT_IPI, which has the same affect as asserting INIT#), which returns it to a wait loop. The result is a possible loss of cache coherency if the off-line processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this erratum.

WORKAROUND: Operating system developers should take care to execute a WBINVD instruction before the AP is taken off-line using an INIT_IPI.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

4. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are **disabled** (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution). Normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register (s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If

a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers before they become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

5. Double ECC Error on Read May Result in BINIT#

PROBLEM: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium II Xeon processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

IMPLICATION: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

WORKAROUND: Though the ability to drive BINIT# can be disabled in the Pentium II Xeon processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

6. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum

can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting a NOP instruction between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

7. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

8. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium II Xeon processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.
IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and SHUTDOWN of the processor.
WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

9. Branch Traps Do Not Function If BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.
IMPLICATION: The branch traps and branch trace message debugging features cannot be used together.

WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

10. Checker BIST Failure in FRC Mode Not Signaled

PROBLEM: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.
IMPLICATION: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.
WORKAROUND: For successful detection of BIST failure in the checker of an FRC pair, use the FRCERR signal, instead of IERR#.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

11. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

PROBLEM: If a pair of Pentium II Xeon processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter SHUTDOWN. The next bus transaction from the master will then result in the assertion of FRCERR.
IMPLICATION: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the behavior of the system specific error recovery mechanisms.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

12. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

IMPLICATION: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter SHUTDOWN. Therefore, leaving MCEs disabled may not improve overall system behavior.
WORKAROUND: No workaround which would guarantee successful MCE handler execution under this condition has been identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

13. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.
IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

14. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data,

the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

IMPLICATION: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

15. A20M# May Be Inverted After Returning from SMM and Reset

PROBLEM: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of non-volatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

IMPLICATION: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

WORKAROUND: Software should save the A20M# signal state in non-volatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

16. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP **after** the return value is pushed on the stack, not the value in the ESP register **before** the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an **indirect** call using ESP (e.g., CALL [ESP]). The saved version of ESP should then be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

17. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

PROBLEM: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.
- Other processors, meanwhile, issue back to back accesses to that same address on the bus.

IMPLICATION: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's

cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.
WORKAROUND: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a non-homogenous memory model.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

18. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

PROBLEM: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Pentium II Xeon processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.
IMPLICATION: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.
WORKAROUND: None identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

19. Memory Type Field Undefined for Non-Memory Bus Transactions

PROBLEM: The Memory Type field for non-memory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for non-memory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.
IMPLICATION: Bus agents may decode a non-UC memory type for non-memory bus transactions.
WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

20. Infinite Snoop Stall During L2 Initialization of MP Systems

PROBLEM: It is possible for snoop traffic generated on the system bus while a processor is executing its L2 cache initialization routine to cause the initializing processor to hang.
IMPLICATION: An MP system which does not suppress snoop traffic while L2 caches are being initialized may hang during this initialization sequence.
WORKAROUND: System BIOS can create an execution environment which allows processors to initialize their L2 caches without the system generating any snoop traffic on the bus.
Below is a pseudo-code fragment, designed explicitly for a 4 processor system, that uses a serial algorithm to initialize each processor's L2 cache:

```
Suppress_all_I/O_traffic()
 K = 0;
 while (K <= 3)
 {
 /* Obtain current value of K. This forces both Temp and K into */
 /* the L1 cache. Note that Temp could also be maintained in a */
 /* general purpose register. */

 Temp = K;
 Wait_until_all_processors_are_signed_in_at_barrier()
 if ( logical_proc_APIC_id == K ) {
 {
 wait_10_usecs_delay_loop(); /* this time delay, required */
 /* in the worst case, allows */
 /* the barrier semaphore to */
 /* settle to shared state. */
 Initialize L2 cache
 K++
 }
 else
 while (Temp == K);
 }
 }
```

This algorithm prevents bus snoop traffic from the other processors, which would otherwise cause the initializing processor to hang. The algorithm assumes that the L1 cache is enabled (the Temp and K variables must be cached by each processor). Also, the Memory Type Range Register (MTRR) for the data segment must be set to WB (writeback) memory type.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

21. FP Data Operand Pointer May Not be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be non-zero after power on or Reset.

IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.

WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

22. Premature Execution of Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur in any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM = 1 (Emulation bit set), a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur, it is possible that the load portion of the instruction will have executed prior to the exception handler being entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

WORKAROUND: Code which performs loads from memory that has side-effects can effectively work around this behavior by using simple integer-based load instructions when accessing side-effect memory, and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

23. EFLAGS Discrepancy on Page Fault After Multiprocessor TLB Shootdown

PROBLEM: This erratum may occur when the Pentium II Xeon processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

IMPLICATION: This scenario may only occur on a multiprocessor platform running an operating system that performs “lazy” TLB shootdowns. The memory image of the EFLAGS register on the page fault handler’s stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

WORKAROUND: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

24. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Pentium II Xeon processor executes a read-modify-write arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.
IMPLICATION: If the memory targeted for the RMW instruction has no side effects, then the memory location will simply be read twice with no additional implications. If, however, the load targets a memory region that has side effects, multiple occurrences of the initial load may lead to unpredictable system behavior.
WORKAROUND: IHVs and ISVs who write device drivers for custom hardware that may have a side effect style of design should use simple loads and simple stores to transfer data to and from the device.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

25. Test Pin Must Be High During Power Up

PROBLEM: The Pentium II Xeon processor core uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, pin A23 of the Pentium II Xeon processor (TEST_VCC_CORE_A23), if at a low voltage level when the core power supply comes up, will cause the processor to enter an invalid test state.
IMPLICATION: If this erratum occurs, the system will fail to power up successfully.
WORKAROUND: Ensure that this pin is pulled up using the core voltage supply. As this pin was previously named TEST_VCC_25_A23, some baseboards may need to tie this pin to 2.5 V. Such baseboards should use a 100 kΩ resistor to ensure proper sequencing on this pin (the internal pull-up will keep the signal from being asserted during power up). Alternatively, ensure that the 2.5 V power supply ($V_{CC2.5}$) reaches a valid level prior to the core voltage supply (V_{CCCORE}).
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

26. Processor Information ROM Uses WP Pin

PROBLEM: The Pentium II Xeon processor contains two memory regions addressable by an SMBus master in a system. One is the Processor Information ROM, which contains

Intel data as defined in the specification, and is intended to be write protected. The other is a Scratch EEPROM which may be defined by the OEM for system use. The Scratch EEPROM can be write-protected by asserting the WP pin on the Pentium II Xeon processor (pin B148). Deasserting this pin allows the Scratch EEPROM to be written using the SMBus. Due to this erratum, deasserting this pin will also allow the Processor Information ROM to be written if it is addressed instead of the Scratch EEPROM.

IMPLICATION: Care must be taken to address only the Scratch EEPROM when writing data using the SMBus. The Processor Information ROM may be overwritten with incorrect information due to this erratum.

WORKAROUND: None identified.

STATUS: For the processor part numbers affected see the Pentium II Xeon Processor Identification and Packaging Information Table in the General Information section.

27. Intervening Writeback May Occur During Split-Lock

PROBLEM: During a transaction which has the LOCK# signal asserted and crosses a cache-line boundary (a.k.a. a split-lock transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a split-lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations can occur. This erratum is, however, a violation of the split-lock protocol.

IMPLICATION: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that split-lock transactions may only consist of a read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

WORKAROUND: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a split-lock transaction.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

28. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the

MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.
IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.
WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

29. Cache Access While Changing BBL_CR_CTL3 Configuration May Cause Hang

PROBLEM: The Model Specific Register (MSR) at address 11E, named BBL_CR_CTL3, is used to configure the core-to-L2 cache interface in the Pentium II Xeon processor. If, during the cache configuration process, a write to BBL_CR_CTL3 occurs which changes the mode of operation of the Pentium II Xeon processor's L2 cache components, and a simultaneous access occurs to cacheable space (such as an L1 cache snoop), a hang condition may result.
IMPLICATION: If caching is enabled and cache snooping occurs during L2 configuration, the system may hang.
WORKAROUND: None identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

30. Thermal Sensor May Assert SMBALERT# Incorrectly

PROBLEM: The Pentium II Xeon processor has a thermal sensor that monitors the processor core's temperature. The thermal sensor asserts SMBALERT# if the processor temperature exceeds the temperature limits set in the Alarm Threshold Registers (T_{HIGH}, T_{LOW}). It also sets the corresponding Status Register bits to identify the cause of the interrupt. Figure 1 gives one example of the how the SMBALERT# signal could be used in a system.

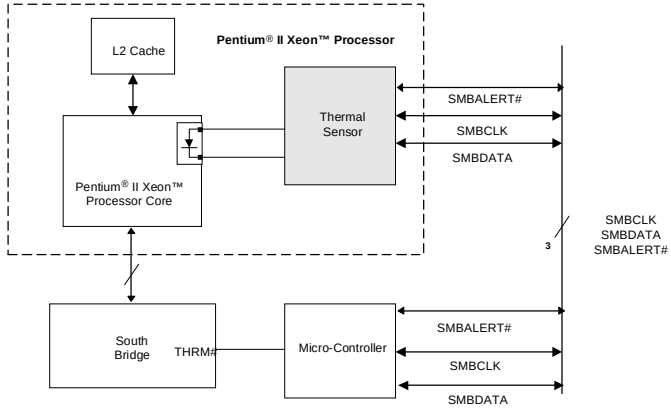

Figure 1. An Example of Micro-controller Driven Thermal Management

Under the conditions described below, the thermal sensor incorrectly generates the SMBALERT# interrupt. **All** of the following conditions must be met to trigger a false interrupt:

1. The thermal sensor must be in auto-convert mode.
2. The absolute value of the difference between the current temperature reading and the T_{HIGH} or T_{LOW} limit value must be less than or equal to 8°C.
3. The current temperature reading must be different from the previous reading.

With a false assertion of SMBALERT#, the corresponding bit in the Status Register (R_{HIGH}, and R_{LOW}) also will be incorrect.

IMPLICATION: There is no system impact from this erratum if temperature polling is used for processor thermal management. If the SMABLERT# interrupt is employed to manage processor thermal sensing, then servicing the false interrupt may result in premature system action depending on the software and hardware implementations used. The rate of the false interrupts is less than the auto-convert rate of the thermal sensor.

WORKAROUND: Three different (mutually exclusive) workarounds are possible:

1. Before servicing an interrupt from the thermal sensor, read and compare the processor thermal reading with the threshold limits (T_{HIGH} or T_{LOW}). Figures 2 and 3 provide basic flowcharts for the implementation of this workaround in an interrupt driven system.
2. If the firmware implemented polls the Status Register only, then before taking any action, re-read the temperature register and do a comparison with the alarm

threshold limits (T_{HIGH} or T_{LOW}) to determine if the value is actually still within the temperature window.

3. Use a temperature polling scheme to monitor the processor temperature.

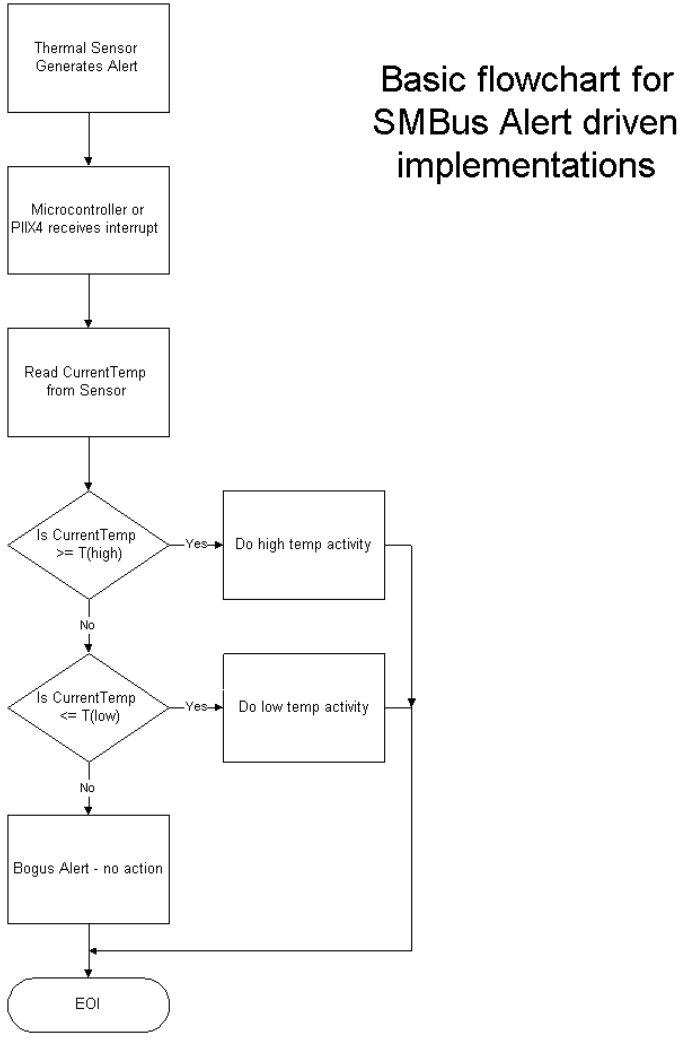

Figure 2. Workaround Flowchart: SMBALERT#-Driven System

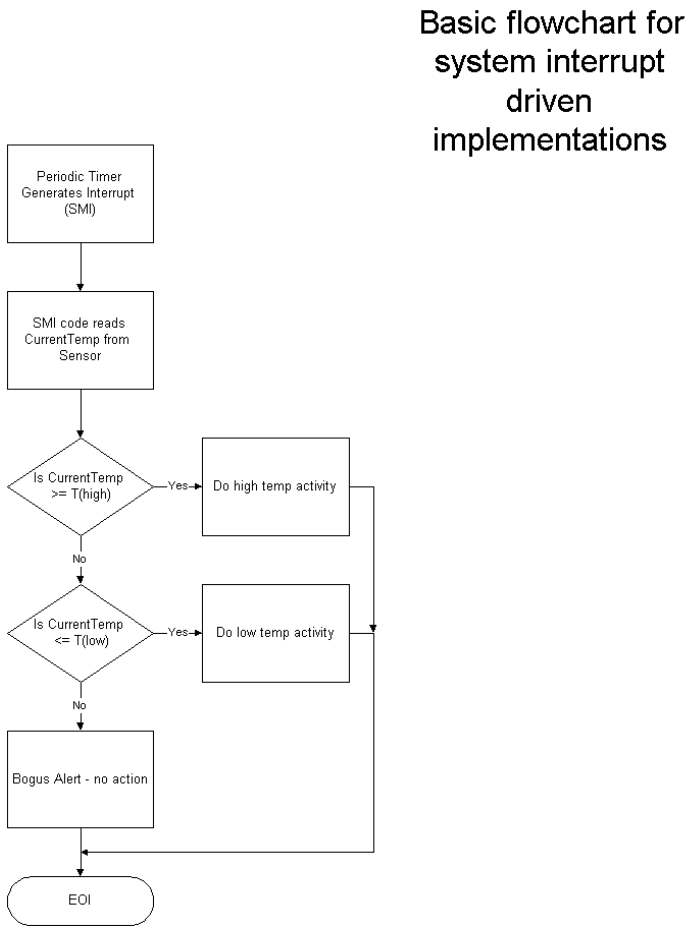

Figure 3. Workaround Flowchart: SMI#-Driven System

STATUS: For the processor part numbers affected see the Pentium II Xeon Processor Identification and Packaging Information Table in the General Information section.

31. MOVD Following Zeroing Instruction Can Cause Incorrect Result

PROBLEM: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,

2. A value is moved with sign extension into the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX™ technology register. Only the MMX technology register is affected by this erratum. The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address>
   or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address>
   or CBW
3. MOVD MM0, EAX

Note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW instruction is specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX or CBW instruction.

IMPLICATION: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

WORKAROUND: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/CBW instruction and the MOVD instruction as in the example below:

   ```
   XOR EAX, EAX (or SUB EAX, EAX)
   MOVSX AX, BL (or other MOVSX or CBW instruction)
   *MOV EAX, EAX
   MOVD MM0, EAX
   ```

   *Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

32. Top 4 PAT Entries Not Usable With Mode B or Mode C Paging

PROBLEM: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Pentium II Xeon processor. However, in Mode B or Mode C paging, the top four entries do not function correctly for 4-Kbyte pages. Specifically, bit 7 of page table entries which translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE = 1) are enabled, the processor forces this bit to zero when determining the memory type, regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).
IMPLICATION: Only the lower four PAT entries are useful for 4-Kbyte translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may also be used for large pages in Mode B or C paging.
WORKAROUND: None identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

33. MOV with Debug Register Causes Debug Exception

PROBLEM: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel*

Architecture Software Developer's Manual, Volume 3: System Programming Guide, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.
IMPLICATION: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.
WORKAROUND: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

PROBLEM: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter SHUTDOWN after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Pentium II Xeon processor to enter SHUTDOWN.
IMPLICATION: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.
WORKAROUND: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2AP. 2-way MP Systems May Hang Due to Catastrophic Errors During BSP Determination

PROBLEM: In 2-way MP systems, a catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.

IMPLICATION: 2-way MP systems may hang during boot due to a catastrophic error. This erratum has not been observed to date in a typical commercial system, but was found during focused system testing using a grounded APIC data bus.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

PROBLEM: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC, (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Pentium II Xeon processor despite the attempt to mask it via the LVT.
IMPLICATION: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.
WORKAROUND: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Pentium® II Xeon™ Processor at 400 MHz* datasheet and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Specification Clarifications will be incorporated into a future version of the appropriate Pentium II Xeon processor documentation.

1. Writes to WC Memory

Section 9.3 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, identifies that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses." This sentence should state that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses. The writes may be delayed until the next occurrence of a buffer or processor serialization event, e.g., CPUID execution, a read or write to uncached memory, interrupt occurrence, locked instruction execution, etc., if the WC buffer is partially filled."

2. Multiple Processor Protocol and Restrictions

Section 7.6.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, contains inconsistencies which will be clarified as follows:

7.5.2 Protocol Requirements and Restrictions

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided on all systems based on P6 family processors (excluding mobile processors and modules).
- All interrupt mechanisms must be disabled for the duration of the MP protocol algorithm, including the window of time between the assertion of INIT# or receipt of an INIT IPI by the application processors and the receipt of a STARTUP IPI by the application processors. That is, requests generated by interrupting devices must not be seen by the local APIC unit (on board the processor) until the completion of the algorithm. Failure to disable the interrupt mechanisms may result in processor shutdown.
- The MP protocol should be initiated only after a hardware reset. After completion of the protocol algorithm, a flag is set in the APIC base MSR of the BSP (APIC_BASE.BSP) to indicate that it is the BSP. This flag is cleared for all other processors. If a processor or the system is subject to an INIT sequence (either through the INIT# pin or an INIT IPI), then the MP protocol is not re-executed.

Instead, each processor examines its BSP flag to determine whether the processor should boot or wait for a STARTUP IPI.

3. Critical Sequence of Events During a Page Fault Exception

Section 3.6.4 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, will be clarified as follows:
If the processor generates a page-fault exception, the operating system must carry out the following operations in this order:

1. Copy the page from disk storage into physical memory if needed.
2. Load the page address into the page-table or page-directory entry and set its present flag. Other bits, such as the dirty and accessed bits, may also be set at this time.
3. Invalidate the current page table entry in the TLB (see Section 3.7., "Translation Lookaside Buffers (TLBs)" for a discussion of TLBs and how to invalidate them).
4. Return from the page fault handler to restart the interrupted program or task.

4. Performance-Monitoring Counter Issues

The following table replaces Table A-1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. The only changes to this new table are enhanced descriptions of the events counted.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_ MEM_ REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only external memory loads and stores, but also internal retries. Note: 80-bit floating-point accesses are double counted, since they are decomposed into a 16-bit exponent load and a 64-bit mantissa load. Memory accesses are only counted when they are actually performed. E.g., a load that gets squashed because a previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once. Does not include I/O accesses, or other non-memory accesses.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	46H	DCU_M_ LINES_IN	00H	Number of M state lines allocated in the DCU.	
	47H	DCU_M_ LINES_ OUT	00H	Number of M state lines evicted from the DCU. This includes evictions via snoop HITM, intervention or replacement.	
	48H	DCU_ MISS_ OUT-STAND-ING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are considered. Uncacheable requests are excluded. RFOs are counted as well as line fills, invalidates, and stores.	An access that also misses the L2 is short-changed by 2 cycles. (i.e., if count is N cycles, should be N+2 cycles.) Subsequent loads to the same cache line will not result in any additional counts. Count value not precise, but still useful.
Instruc tion Fetch Unit (IFU)	80H	IFU_ IFETCH	00H	Number of instruction fetches, both cacheable and uncacheable. Including UC fetches.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	85H	ITLB_ MISS	00H	Number of ITLB misses.	
	86H	IFU_ MEM_ STALL	00H	Number of cycles instruction fetch is stalled, for any reason. Includes IFU cache misses, ITLB misses, ITLB faults and other minor stalls.	
	87H	ILD_ STALL	00H	Number of cycles that the instruction length decoder is stalled.	
L2 Cache 1	28H	L2_ IFETCH	MESI 0FH	Number of L2 instruction fetches. This event indicates that a normal instruction fetch was received by the L2. The count includes only L2 cacheable instruction fetches; it does not include UC instruction fetches. It does not include ITLB miss accesses.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2. Specifically, it indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable store memory accesses; it does not include I/O accesses, other non-memory accesses, or memory accesses like UC/WT stores. It includes TLB miss memory accesses.	
	24H	L2_LINES_IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_OUT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_LINES_INM	00H	Number of modified lines allocated in the L2.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	2EH	L2_ RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_ BUSY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_ BUSY_ RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	
External Bus Logic (EBL) 2	62H	BUS_ DRDY_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which DRDY# is asserted. Essentially, utilization of the external system data bus.	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#. Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.
	63H	BUS_ LOCK_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus.	Always counts in processor clocks.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	65H	BUS_ TRAN_ BRD	00H (Self) 20H (Any)	Number of burst read transactions.	
	66H	BUS_ TRAN_ RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_ TRANS_W B	00H (Self) 20H (Any)	Number of completed write back transactions.	
	68H	BUS_ TRAN_ IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	
	69H	BUS_ TRAN_ INVAL	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_ TRAN_ PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_ TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_ TRANS_ IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_ TRAN_ DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	70H	BUS_ TRAN_ ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles, etc.	
	6FH	BUS_ TRAN_ MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	
	64H	BUS_ DATA_ RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR _DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	7BH	BUS_ HITM_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7EH	BUS_ SNOOP_ STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	10H	FP_COMP_ OPS_ EXE	00H	Number of computational floating-point operations executed. The number of FADD, FSUB, FCOM, FMUL, integer MUL and IMUL, FDIV, FPREM, FSQRTS, integer DIV and IDIV instructions. Note this is not the number of cycles, but the number of operations. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	Counter 0 only
	11H	FP_ ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	Number of multiplies. Note: includes integer and well FP multiplies and is speculative.	Counter 1 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	14H	CYCLES_ DIV_ BUSY	00H	Number of cycles that the divider is busy, and cannot accept new divides. Note: includes integer and FP divides, FPREM, FPSQRT, etc., and is speculative.	Counter 0 only
Mem-ory Or-dering	03H	LD_ BLOCKS	00H	Number of store buffer blocks. Includes counts caused by preceding stores whose addresses are unknown, preceding stores whose addresses are known to conflict, but whose data is unknown and preceding stores that conflicts with the load, but which incompletely overlap the load.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	05H	MIS-ALIGN_ MEM_ REF	00H	Number of misaligned data memory references. Incremented by 1 every cycle during which either the processor load or store pipeline dispatches a misaligned uop. Counting is performed if its the first half or second half, or if it is blocked, squashed or misses. Note in this context misaligned means crossing a 64 bit boundary.	It should be noted that MISALIGN_MEM_REF is only an approximation, to the true number of misaligned memory references. The value returned is roughly proportional to the number of misaligned memory accesses, i.e., the size of the problem.
Instruction De-coding and Retire-ment	C0H	INST_ RETIRED	00H	Number of instructions retired.	A hardware interrupt received during/after the last iteration of the REP STOS flow causes the counter to undercount by 1 instruction.
	C2H	UOPS_ RETIRED	00H	Number of uops retired.	
	D0H	INST_ DECOD-ER	00H	Number of instructions decoded.	
Inter-rupts	C8H	HW_INT_ RX	00H	Number of hardware interrupts received.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	C7H	CYCLES_ INT_ PENDING_ AND_ MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Branches	C4H	BR_INST_ RETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_ PRED_ RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_ TAKEN_ RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_ PRED_ TAKEN_ RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_ DECOD-ED	00H	Number of branch instructions decoded.	
	E2H	BTB_ MISSES	00H	Number of branches that for which the BTB did not produce a prediction.	
	E4H	BR_ BOGUS	00H	Number of bogus branches.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Stalls	A2H	RE-SOURCE_ STALLS	00H	Incremented by one during every cycle that there is a resource related stall. Includes register renaming buffer entries, memory buffer entries. Does not include stalls due to bus queue full, too many cache misses, etc. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, e.g., if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	
	D2H	PARTIAL_ RAT_ STALLS	00H	Number of cycles or events for partial stalls. Note Includes flag partial stalls.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Clocks	79H	CPU_CLK_ UN-HALTED	00H	Number of cycles during which the processor is not halted.	

NOTES:

1. Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved. The Pentium® II Xeon™ processor identifies cache states using the "MESI" protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.
2. All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).

5. POP[ESP] With 16-bit Stack Size

In Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, the entry for "POP–Pop a Value from the Stack" includes the following note:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register."

This note is incomplete, and should read as follows:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. For the case of a 16-bit stack where ESP wraps to 0h as a result of the POP instruction, the resulting location of the memory write is processor family specific."

Also, in Section 17.23.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, a new section will be added:

A POP-to-Memory Instruction Which Uses the Stack Pointer (ESP) as a Base Register

For a POP-to-memory instruction that meets the following conditions:

1. The stack segment size is 16 bits,
2. A 32-bit addressing form uses the SIB byte to specify ESP as the base register, and
3. The initial stack pointer is FFFCh (32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0h as a result of the POP operation,

The result of the memory write is processor family specific. For example, in Pentium II, Pentium II Xeon, and Pentium Pro processors the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and i486™ processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 Kbytes), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbytes).

6. Preventing Caching

Section 9.5.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, documents the procedure to prevent the L1 and L2 caches from performing all caching operations. However, this procedure differs from that given in Section 11.11.8, "Multiple-Processor Considerations." The correct procedure that should be used is as follows:

1. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.)
2. Flush all caches using the WBINVD instruction.
3. Disable the MTRRs and set the default memory type to uncached, or set all MTRRs for the uncached memory type (see the discussion of the TYPE field and the E flag in Section 11.11.2.1, "MTRRdefType Register").

The caches must be flushed when the CD flag is cleared to insure system memory coherency. If the caches are not flushed in step 2, cache hits on reads will still occur and data will be read from valid cache lines.

7. Paging Must Be Enabled Before Enabling the Page Global Enable Bit

In Section 2.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, the following line will be added to the text describing the Page Global Enable bit (PGE).

"In addition, the bit must not be enabled before paging is enabled via CR0.PG. Program correctness may be affected by reversing this sequence and processor performance will be impacted."

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Pentium® II Xeon™ Processor at 400 MHz* datasheet and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Documentation Changes will be incorporated into a future version of the appropriate Pentium II Xeon processor documentation.

1. Invalid Arithmetic Operations and Masked Responses to Them Relative to FIST/FISTP Instruction

The *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, Table 7-20 shows "Invalid Arithmetic Operations and the Masked Responses to Them." The table entry corresponding to the FIST/FISTP condition is missing, and is shown below:

Condition	Masked Response
FIST/FISTP instruction when input operand <> MAXINT for destination operand size.	Return MAXNEG to destination operand.

When FIST/FISTP instruction is executed with input operand <> and the destination operand size is MAXINT, the floating-point zero-divide exception will return MAXNEG to the destination operand as its masked response.

2. FIDIV/FIDIVR **m16int** *Description*

Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference,* shows, in the Description column for the FIDIV *m16int* instruction, "Divide ST(0) by *m64int* by ST(0) and store the result in ST(0)", and for the FIDIVR *m16int* instruction, "Divide *m64int* by ST(0) and store the result in ST(0)". In both of these cases, *m64int* should be replaced with *m16int*.

3. PUSH Does Not Pad With Zeros

In Section 4.2.2 of the *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, the last sentence in the first paragraph reads "If a 16-bit value is pushed onto a 32-bit wide stack, the value is automatically padded with zeros out to 32-bits." This sentence should be removed. The PUSH instruction does not pad with zeros.

4. DR7, Bit 10 is Reserved

Section 14.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guid,e* contains Figure 14-1. This figure should show bit 10 of DR7 as "Reserved" instead of "1".

5. Additional States That Are Not Automatically Saved and Restored

The end of Section 11.4.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* lists the registers that are not automatically saved and restored following an SMI or an RSM instruction, respectively. The last two paragraphs should be as follows:

The following state is not automatically saved and restored following an SMI or an RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium® processors), or test registers TR3 through TR7 (for the Pentium and Intel486™ processors).
- The state of the trap controller.
- The Machine-Check Architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The Microcode Update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So, an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor it must also save and restore them.

NOTE

A small subset of the MSRs (such as the time-stamp counter and performance-monitoring counter) are not arbitrarily writeable and therefore cannot be saved and restored. SMM-based power-

down and restoration should only be performed with operating systems that do not use or rely on the values of these registers. Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

6. Cache and TLB Description Correction

In Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Table 3-7 (in the description of the CPUID instruction), the correct description for descriptor value 02h should be as follows:

Descriptor Value	Cache or TLB Description
02H	Instruction TLB: 4M-Byte Pages, **fully** associative, **2** entries

Also, the third bullet after the table should be as follows:

- Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:
  - 01H–A 32-entry instruction TLB (4-way set associative) for mapping 4-Kbytes pages.
  - 02H–A **2**-entry instruction TLB (**fully** associative) for mapping 4-Mbyte pages.
  - 03H–A 64-entry data TLB (4-way set associative) for mapping 4-Kbyte pages.

In Section 9.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Table 9-1, the following corrections should be made:

Cache or Buffer	Characteristics
Instruction TLB (Large Pages)	- P6 family processors: **2** entries, **fully** associative. - Pentium® processor: Uses same TLB as used for 4-Kbyte pages. - Intel486™ processor: None (large pages not supported).
Data TLB (Large Pages)	- P6 family processors: 8 entries, 4-way set associative. - Pentium processor: **8** entries, **4**-way set associative; uses same TLB as used for 4-Kbyte pages in Pentium processors with MMX™ technology. - Intel486 processor: None (large pages not supported).

7. SMRAM State Save Map Contains Documentation Errors

In the *Intel Architecture Software Developer's Manual, Volume 3, System Programming Guide*, Chapter 11, "System Management Mode," Table 11-1 incorrectly documents the SMBASE + Offset for IDT Base and GDT Base for Pentium II Xeon processors.

The storage locations for these parameters are model specific (i.e., they may differ between the Pentium processor, the Pentium Pro processor, Pentium II processor, and other P6 family proliferations). These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.

8. OF and DF of the EFLAGS Register are Mislabeled as System Flags

In the *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, Table 3-7, the Overflow Flag (OF) and Direction Flag (DF) are both incorrectly labeled as System Flags. The Overflow Flag should be labeled as a Status Flag and the Direction Flag should be labeled as a Control Flag.

9. CS:EIP Pushed Onto Stack Prior to Code Segment Limit Check

The *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Section 3.4, contains a detailed definition of the CALL instruction. In this definition, all instances where the instruction pointer is checked to ensure it is within the acceptable code segment limit followed by the CS:EIP register being pushed on the stack are in error. CS:EIP is pushed on the stack prior to the check of the instruction

pointer. This means that in the case of a GP#(0) being generated due to an out-of-range instruction pointer, these values will be present on the stack.

10. Corrections to Opcode Maps

In Appendix A, "Opcode Map," in the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, one and two byte opcode maps are documented. The following tables are intended to replace those tables in their entirety:

Table A-1. One-Byte Opcode Map[1]

	0	1	2	3	4	5	6	7
0	ADD						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	ES	ES
1	ADC						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	SS	SS
2	AND							DAA
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=ES	
3	XOR							AAA
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=SS	
4	INC general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	PUSH general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSHA/ PUSHAD	POPA/ POPAD	BOUND	ARPL			Operand	Address
			Gv,Ma	Ew,Gw	=FS	=GS	Size	Size
7	Short-displacement jump on condition (Jb)							
	JO	JNO	JB/JNAE/JC	JNB/ JAE/JNC	JZ/JE	JNZ/ JNE	JBE/ JNA	JNBE/ JA
8	Imm Group 1[2]			Imm Group 1[2]	TEST		XCHG	
	Eb,Ib	Ev,Iv	Ev,Ib	Eb,lb	Eb,Gb	Ev,Gv	Eb,Gb	Ev,Gv
9	NOP	XCHG word or double-word register with eAX						
		eCX	eDX	eBX	eSP	eBP	eSI	eDI
A	MOV				MOVSB	MOVSW	CMPSB	CMPSW
	AL,Ob	eAX,Ov	Ob,AL	Ov,eAX	Xb,Yb	Xv,Yv	Xb,Yb	Xv,Yv
B	MOV immediate byte into byte register							

Table A-1. One-Byte Opcode Map[1] (Contd.)

	AL	CL	DL	BL	AH	CH	DH	BH
C	Shift Group 2[2]		RET near		LES	LDS	MOV	
	Eb,Ib	Ev,Ib	Iw		Gv,Mp	Gv,Mp	Eb,Ib	Ev,Iv
D	Shift Group 2[2]				AAM	AAD		XLAT/ XLATB
	Eb,1	Ev,1	Eb,CL	Ev,CL				
E	LOOPNE/ LOOPNZ	LOOPE/LO OPZ	LOOP	JCXZ/ JECXZ	IN		OUT	
	Jb	Jb	Jb	Jb	AL,Ib	eAX,Ib	Ib,AL	Ib,eAX
F	LOCK		REPNE	REP/ REPE	HLT	CMC	Unary Group 32	
							Eb	Ev
	8	9	A	B	C	D	E	F
0	OR						PUSH	2-byte
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	CS	Escape
1	SBB						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	DS	DS
2	SUB							DAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv		
3	CMP							AAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=DS	
4	DEC General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	POP Into General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSH	IMUL	PUSH	IMUL	INSB	INSW/D	OUTSB	OUTSW/D
	lv	Gv,Ev,lv	lb	Gv,Ev,lb	Yb,DX	Yv,DX	Dx,Xb	DX,Xv
7	Short-Displacement Jump on Condition (Jb)							
	JS	JNS	JP/JPE	JNP/JPO	JL/JNGE	JNL/JGE	JLE/JNG	JNLE/JG
8	MOV					LEA	MOV	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	Ew,Sw	Gv,M	Sw,Ew	Ev
9	CBW	CWD/ CDQ	CALL	FWAIT	PUSHF/ PUSHFD	POPF/ POPFD	SAHF	LAHF
			Ap		Fv	Fv		
A	TEST		STOS/ STOSB	STOS/STO SW/STOTS D	LODSB	LODSW/LO DSD	SCAS/ SCACSB	SCASW/ SCASD/ SCAS

Table A-1. One-Byte Opcode Map[1] (Contd.)

	AL,Ib	eAX,Iv	Yb,AL	Yv,eAX	AL,Xb	eAX,Xv	AL,Yb	eAX,Yv
B	MOV Immediate Word or Double Into Word or Double Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
C	ENTER	LEAVE	RET far	RET far	INT 3	INT	INTO	IRET
	Iw, Ib		Iw			Ib		
D	ESC (Escape to Coprocessor Instruction Set)							
E	CALL	JMP			IN		OUT	
	Jv	Jv	Ap	Jb	AL,DX	eAX,DX	DX,AL	DX,eAX
F	CLC	STC	CLI	STI	CLD	STD	Group 4[2]	Group 5[2]

NOTES:

1. All blanks in the opcode map are reserved and should not be used. Do not depend on the operation of these undefined opcodes.
2. Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (see Section A.4.).

Table A-2. Two Byte Opcode Map (First byte is 0Fh)[1]

	0	1	2	3	4	5	6	7
0	Group 6[2]	Group 7[2]	LAR	LSL			CLTS	
			Gv,Ew	Gv,Ew				
1								
2	MOV							
	Rd,Cd	Rd,Cd	Cd,Rd	Dd,Rd				
3	WRMSR	RDTSC	RDMSR	RDPMC				
4	CMOVO	CMOVNO	CMOVB/ CMOVC/ CMOVNAE	CMOVAE/ CMOVNB/ CMOVNC	CMOVE/ CMOVZ	CMOVNE/ CMOVNZ	CMOVBE/ CMOVNA	CMOVA/ CMOVNBE
	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev
5								
6	PUNPCKLBW	PUNPCKLWD	PUNPCKLDQ	PACKSSDW	PCMPGTB	PCMPGTW	PCMPGTD	PACKUSWB

Table A-2. Two Byte Opcode Map (First byte is 0Fh)[1] (Contd.)

	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd
7	Group A[2]				PCMPEQB	PCMPEQW	PCMPEQD	EMMS
		PSHIMW[3]	PSHIMD[3]	PSHIMQ[3]	Pq, Qd	Pq, Qd	Pq, Qd	
8	Long-Displacement Jump on Condition (Jv)							
	JO	JNO	JB/JNAE/JC	JAE/JNB/JNC	JE/JZ	JNE/JNZ	JBE/JNA	JA/JNBE
9	Byte Set on condition (Eb)							
	SETO	SETNO	SETB/SETC/SETNA	SETAE/SETNB/SETNC	SETE/SETG/SETZ	SETNE/SETNZ	SETBE/SETNA	SETA/SETNBE
A	PUSH	POP	CPUID	BT	SHLD	SHLD		
	FS	FS	Ev,Gv	Ev,Gv,Ib	Ev,Gv,CL			
B	CMPXCHG	CMPXCHG	LSS	BTR	LFS	LGS	MOVZX	
	Eb,Gb	Ev,Gv	Mp	Ev,Gv	Mp	Mp	Gv,Eb	Gv,Ew
C	XADD	XADD						Group 9[2]
	Eb,Gb	Ev,Gv						
D		PSRLW	PSRLD	PSRLQ		PMULLW		
		Pq, Qd	Pq, Qd	Pq, Qd		Pq, Qd		
E		PSRAW	PSRAD			PMULHW		
		Pq, Qd	Pq, Qd			Pq, Qd		
F		PSLLW	PSLLD	PSLLQ		PMADDWD		
		Pq, Qd	Pq, Qd	Pq, Qd		Pq, Qd		
	8	9	A	B	C	D	E	F
0	INVD	WBINVD		UD24				
1								
2								
3								

Table A-2. Two Byte Opcode Map (First byte is 0Fh)[1] (Contd.)

4	CMOVS	CMOVNS	CMOVP/ CMOVPE	CMOVNP/C MOVPO	CMOVL/ CMOVN GE	CMOVGE/ CMOVNL	CMOVLE/C MOVNG	CMOVG/ CMOVNLE
	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev
5								
6	PUNPCK HBW	PUNPCKH WD	PUNPCK HDQ	PACKSSD W			MOVD	MOVQ
	Pq,Qd	Pq,Qd	Pq,Qd	Pq,Qd			Pd,Ed	Pq,Qq
7							MOVD	MOVQ
							Ed,Pd	Qq,Pq
8	Long-Displacement Jump on Condition (Jv)							
	JS	JNS	JP/JPE	JNP/JPO	JL/JNGE	JNL/JGE	JLE/JNG	JNLE/JG
	Byte set on condition (Eb)							
9	SETS	SETNS	SETP/ SETPE	SETNP/ SETPO	SETL/ SETNGE	SETNL/ SETGE	SETLE/ SETNG	SETNLE
	Eb	Eb	Eb	Eb	Eb	Eb	Eb	Eb
A	PUSH	POP	RSM	BTS	SHRD	SHRD		IMUL
	GS	GS		Ev,Gv	Ev,Gv,Ib	Ev,Gv,CL		Gv,Ev
B		Invalid Opcode[4]	Group 8[2]	BTC	BSF	BSR	MOVSX	
			Ev,lb	Ev,Gv	Gv,Ev	Gv,Ev	Gv,Eb	Gv,Ew
C	BSWAP							
	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
D	PSUBUS B	PSUBUSW		PAND	PADDUS B	PADDUSW		PANDN
	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq
E	PSUBSB	PSUBSW		POR	PADDSB	PADDSW		PXOR
	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq
F	PSUBB	PSUBW	PSUBD		PADDB	PADDW	PADDD	
	Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq	

NOTES:

1. All blanks in the opcode map are reserved and should not be used. Do not depend on the operation of these undefined opcodes.
2. Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (see Section A.4.).
3. These abbreviations are not actual mnemonics. When shifting by immediate shift counts, the PSHIMD mnemonic represents the PSLLD, PSRAD, and PSRLD instructions, PSHIMW represents the PSLLW, PSRAW, and PSRLW instructions, and PSHIMQ represents the PSLLQ and PSRLQ instructions. The instructions that shift by immediate counts are differentiated by the ModR/M bytes (see Section A.4).
4. Use the 0F0Bh opcode (UD2 instruction) or the 0FB9h opcode when deliberately trying to generate an invalid opcode exception (#UD).

11. MP Initialization Protocol Algorithm Correction

In Section 7.6.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, the algorithm for MP initialization is defined. It is stated "the APIC hardware observes the BNR# (block next request) and BPRI# (priority agent bus request) pins to guarantee that the initial BIPI is not issued on the APIC bus until the BIST sequence is complete for all processors in the system." This is not correct. Only the observation of BNR# is required for the APIC hardware to proceed.

12. Interrupt 13-General Protection Exception (#GP)

In Section 5.12 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, a description of the exception interrupts is provided. In the description section of Interrupt 13-General Protection Exception (#GP), the last bullet applies if the PAE and/or PSE flags are set, rather than just the PAE flag as reported in the documentation.

UNITED STATES, Intel Corporation
2200 Mission College Blvd., P.O. Box 58119, Santa Clara, CA 95052-8119
Tel: +1 408 765-8080

JAPAN, Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi, Ibaraki-ken 300-26
Tel: + 81-29847-8522

FRANCE, Intel Corporation S.A.R.L.
1, Quai de Grenelle, 75015 Paris
Tel: +33 1-45717171

UNITED KINGDOM, Intel Corporation (U.K.) Ltd.
Pipers Way, Swindon, Wiltshire, England SN3 1RJ
Tel: +44 1-793-641440

GERMANY, Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen/ Muenchen
Tel: +49 89/99143-0

HONG KONG, Intel Semiconductor Ltd.
32/F Two Pacific Place, 88 Queensway, Central
Tel: +852 2844-4555

CANADA, Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8
Tel: +416 675-2438